FOUNDATION-0.6: Agent Registry - PART 2 (Execution & Testing)

🎯 CONTEXT

Phase: FOUNDATION (Week 1 - Day 3 Morning)

Component: Agent Registry - Execution & Validation

Estimated Time: 10 min execution + 15 min testing

Complexity: MEDIUM

Risk Level: LOW

Files: Part 2 of 2 (Execution, testing, validation)

📦 PREREQUISITES

Must Have Completed:

- ✅ **PART 1** - All code files created
- ✅ **P-02** - Orchestrator skeleton
- ✅ **0.2a-0.2e** - PostgreSQL schemas
- ✅ **0.3** - ClickHouse
- ✅ **0.4** - Qdrant

Verify PART 1 Files Exist:

```bash
cd ~/optiinfra

# Check Go files
ls -lh services/orchestrator/internal/registry/models.go
ls -lh services/orchestrator/internal/registry/registry.go
ls -lh services/orchestrator/internal/registry/handlers.go
ls -lh services/orchestrator/cmd/server/main.go

# Check Python files
ls -lh shared/orchestrator/registration.py
ls -lh shared/orchestrator/__init__.py

# Expected: All files present
```

🚀 STEP-BY-STEP EXECUTION

Step 1: Create Go Directory Structure

```bash
cd ~/optiinfra/services/orchestrator

# Create internal directory structure
mkdir -p internal/registry
mkdir -p cmd/server

# Verify structure
tree -L 3
# Expected:
# .
# ├── cmd/
# │   └── server/
# └── internal/
#     └── registry/
```

Validation:

```bash
# Check directories exist
[ -d "internal/registry" ] && echo "✅ internal/registry/ created" || echo "❌ Missing"
[ -d "cmd/server" ] && echo "✅ cmd/server/ created" || echo "❌ Missing"
```

Step 2: Create Go Model Files

```bash
```

```
cd ~/optiinfra/services/orchestrator/internal/registry

# Create models.go (copy from PART 1, FILE 1)
cat > models.go << 'EOF'
package registry

import (
  "time"
)
[... COPY ENTIRE MODELS.GO FROM PART 1, FILE 1 ...]
EOF

# Verify file
ls -lh models.go
wc -l models.go
# Expected: ~150 lines
```

Validation:

```bash
# Check file size
FILE_SIZE=$(wc -l < models.go)
if [ "$FILE_SIZE" -gt 100 ]; then
    echo "✅ models.go created ($FILE_SIZE lines)"
else
    echo "❌ models.go too small ($FILE_SIZE lines)"
fi

# Check for key types
grep -q "type Agent struct" models.go && echo "✅ Agent struct found" || echo "❌ Missing"
grep -q "type AgentType" models.go && echo "✅ AgentType found" || echo "❌ Missing"
```

Step 3: Create Registry Core Logic

```bash
```

```
cd ~/optiinfra/services/orchestrator/internal/registry

# Create registry.go (copy from PART 1, FILE 2)
cat > registry.go << 'EOF'
package registry
[... COPY ENTIRE REGISTRY.GO FROM PART 1, FILE 2 ...]
EOF

# Verify file
ls -lh registry.go
wc -l registry.go
# Expected: ~400 lines
```

Validation:

```bash

# Check file size
FILE_SIZE=$(wc -l < registry.go)
if [ "$FILE_SIZE" -gt 350 ]; then
    echo "✅ registry.go created ($FILE_SIZE lines)"
else
    echo "❌ registry.go too small"
fi

# Check for key functions
grep -q "func NewRegistry" registry.go && echo "✅ NewRegistry found" || echo "❌ Missing"
grep -q "func.*Register" registry.go && echo "✅ Register found" || echo "❌ Missing"
grep -q "func.*Heartbeat" registry.go && echo "✅ Heartbeat found" || echo "❌ Missing"
```

Step 4: Create HTTP Handlers

```bash
```

```bash
cd ~/optiinfra/services/orchestrator/internal/registry

# Create handlers.go (copy from PART 1, FILE 3)
cat > handlers.go << 'EOF'
package registry
[... COPY ENTIRE HANDLERS.GO FROM PART 1, FILE 3 ...]
EOF

# Verify file
ls -lh handlers.go
wc -l handlers.go
# Expected: ~150 lines
```

Step 5: Update Main Server File

```bash
cd ~/optiinfra/services/orchestrator/cmd/server

# Create/update main.go (copy from PART 1, FILE 4)
cat > main.go << 'EOF'
package main
[... COPY ENTIRE MAIN.GO FROM PART 1, FILE 4 ...]
EOF

# Verify file
ls -lh main.go
```

Step 6: Create/Update go.mod

```bash
```

```
cd ~/optiinfra/services/orchestrator

# Create go.mod (copy from PART 1, FILE 5)
cat > go.mod << 'EOF'
module optiinfra/services/orchestrator

go 1.21

require (
  github.com/gin-gonic/gin v1.9.1
  github.com/go-redis/redis/v8 v8.11.5
  github.com/google/uuid v1.5.0
)
EOF

# Download dependencies
go mod tidy
go mod download

# Expected: Dependencies downloaded
```

Validation:

```bash
# Verify dependencies
go list -m all | head -10

# Check if key dependencies present
go list -m github.com/gin-gonic/gin && echo "✅ gin installed" || echo "❌ Missing"
go list -m github.com/go-redis/redis/v8 && echo "✅ redis installed" || echo "❌ Missing"
go list -m github.com/google/uuid && echo "✅ uuid installed" || echo "❌ Missing"
```

Step 7: Create Python Registration Helper

```bash
```

```
cd ~/optiinfra/shared

# Create orchestrator directory
mkdir -p orchestrator

cd orchestrator

# Create registration.py (copy from PART 1, FILE 6)
cat > registration.py << 'EOF'
"""
Agent registration helper for Python agents.
[... COPY ENTIRE REGISTRATION.PY FROM PART 1, FILE 6 ...]
"""
EOF

# Create __init__.py (copy from PART 1, FILE 7)
cat > __init__.py << 'EOF'
"""
Orchestrator client utilities for Python agents.
"""

from shared.orchestrator.registration import AgentRegistration

__all__ = ['AgentRegistration']
EOF

# Verify files
ls -lh registration.py
ls -lh __init__.py
```

Step 8: Build the Orchestrator

```
bash
```

```
cd ~/optiinfra/services/orchestrator

# Build the Go binary
go build -o bin/orchestrator ./cmd/server

# Verify binary created
ls -lh bin/orchestrator
# Expected: Binary file ~10-20MB

# Test binary
./bin/orchestrator --help 2>&1 || echo "Binary runs"
```

Validation:

```bash
# Check if binary was created
if [ -f "bin/orchestrator" ]; then
  echo "✅ Orchestrator binary built successfully"
  ls -lh bin/orchestrator
else
  echo "❌ Build failed - check for errors above"
  exit 1
fi
```

Step 9: Start Redis (if not running)

```bash
```

```
cd ~/optiinfra

# Check if Redis is running
docker ps | grep redis

# If not running, start all services
docker-compose up -d redis

# Wait for Redis
sleep 3

# Test Redis connection
docker exec optiinfra-redis redis-cli ping
# Expected: PONG
```

Step 10: Run the Orchestrator

```bash
cd ~/optiinfra/services/orchestrator

# Set environment variables
export REDIS_ADDR="localhost:6379"
export PORT="8080"

# Run orchestrator
./bin/orchestrator

# Expected output:
# Connected to Redis
# Agent registry started
# Starting orchestrator on port 8080
```

Keep this terminal running! Open a new terminal for testing.

Step 11: Test Health Endpoint

```bash
```

```
# In a NEW terminal
cd ~/optiinfra

# Test health endpoint
curl http://localhost:8080/health

# Expected output:
# {
#   "service": "orchestrator",
#   "status": "healthy",
#   "timestamp": "2025-10-20T..."
# }
```

Validation:

```bash
# Test health with validation
RESPONSE=$(curl -s http://localhost:8080/health)
if echo "$RESPONSE" | grep -q "healthy"; then
    echo "✅ Orchestrator is healthy"
else
    echo "❌ Health check failed"
    echo "Response: $RESPONSE"
fi
```

Step 12: Test Agent Registration (CLI)

```bash
```

```
# Register a test agent via CLI
curl -X POST http://localhost:8080/agents/register \
  -H "Content-Type: application/json" \
  -d '{
    "name": "cost-agent-test",
    "type": "cost",
    "host": "localhost",
    "port": 8001,
    "capabilities": ["spot_migration", "reserved_instances"],
    "version": "1.0.0"
  }'

# Expected output:
# {
#   "agent_id": "550e8400-e29b-41d4-a716-446655440000",
#   "registered_at": "2025-10-20T10:30:00Z",
#   "heartbeat_url": "/agents/550e8400.../heartbeat",
#   "heartbeat_interval_seconds": 30
# }
```

Save the agent_id for next steps!

Step 13: Test Listing Agents

```
bash
```

```bash
# List all registered agents
curl http://localhost:8080/agents

# Expected output:
# {
#   "agents": [
#     {
#       "id": "550e8400...",
#       "name": "cost-agent-test",
#       "type": "cost",
#       "status": "healthy",
#       ...
#     }
#   ],
#   "count": 1
# }
```

Step 14: Test Heartbeat

```
bash

# Send heartbeat (replace AGENT_ID with actual ID from step 12)
AGENT_ID="your-agent-id-here"

curl -X POST http://localhost:8080/agents/$AGENT_ID/heartbeat \
  -H "Content-Type: application/json" \
  -d '{
    "status": "healthy",
    "metadata": {
      "requests_processed": 100,
      "cpu_usage": 45.5
    }
  }'

# Expected output:
# {
#   "received": true,
#   "next_interval_seconds": 30,
#   "timestamp": "2025-10-20T..."
# }
```

Step 15: Test Python Registration Helper

```bash
```

```
cd ~/optiinfra

# Create test script
cat > test_registration.py << 'EOF'
#!/usr/bin/env python3
"""Test agent registration from Python."""

import time
import logging
from shared.orchestrator.registration import AgentRegistration

logging.basicConfig(level=logging.INFO)

def main():
    print("=== Testing Python Agent Registration ===\n")

    # Create registration
    registration = AgentRegistration(
        agent_name="python-test-agent",
        agent_type="cost",
        host="localhost",
        port=9001,
        capabilities=["spot_migration", "right_sizing"],
        orchestrator_url="http://localhost:8080",
        version="1.0.0"
    )

    # Register
    print("1. Registering agent...")
    if registration.register():
        print(f"  ✅ Registered with ID: {registration.agent_id}\n")
    else:
        print("  ❌ Registration failed")
        return

    # Start heartbeat
    print("2. Starting heartbeat...")
    registration.start_heartbeat()
    print("  ✅ Heartbeat started\n")

    # Wait a bit
    print("3. Waiting 35 seconds (will send 1 heartbeat)...")
    time.sleep(35)
    # Check if still registered
    print("\n4. Agent should still be registered")

    # Unregister
    print("\n5. Unregistering...")
    registration.unregister()
    print("   ✅ Unregistered\n")

    print("=== Test Complete ===")

if __name__ == "__main__":
    main()
EOF

# Make executable
chmod +x test_registration.py

# Run test
python test_registration.py
```

Expected Output:

```
=== Testing Python Agent Registration ===

1. Registering agent...
   ✅ Registered with ID: abc12345-...

2. Starting heartbeat...
   ✅ Heartbeat started

3. Waiting 35 seconds (will send 1 heartbeat)...

4. Agent should still be registered

5. Unregistering...
   ✅ Unregistered

=== Test Complete ===
```

Step 16: Test Agent Discovery by Type

```bash
# Get all cost agents
curl http://localhost:8080/agents/type/cost

# Expected: List of cost agents

# Get performance agents (should be empty)
curl http://localhost:8080/agents/type/performance

# Expected: {"agents": [], "count": 0}
```

✅ COMPREHENSIVE VERIFICATION

Run this complete verification script:

```bash
```

```
cd ~/optiinfra

cat > verify_registry.sh << 'EOF'
#!/bin/bash

echo "======================================================================"
echo "FOUNDATION-0.6 COMPREHENSIVE VERIFICATION"
echo "======================================================================"

# Colors
GREEN='\033[0;32m'
RED='\033[0;31m'
NC='\033[0m' # No Color

# Test counter
PASSED=0
FAILED=0

# Test function
test() {
  if [ $? -eq 0 ]; then
    echo -e "${GREEN}✅ $1${NC}"
    ((PASSED++))
  else
    echo -e "${RED}❌ $1${NC}"
    ((FAILED++))
  fi
}

# 1. Check if orchestrator is running
echo ""
echo "1. ORCHESTRATOR HEALTH"
curl -s http://localhost:8080/health | grep -q "healthy"
test "Orchestrator is healthy"

# 2. Register test agent
echo ""
echo "2. AGENT REGISTRATION"
RESPONSE=$(curl -s -X POST http://localhost:8080/agents/register \
 -H "Content-Type: application/json" \
 -d '{
  "name": "verify-test-agent",
  "type": "cost",
  "host": "localhost",
  "port": 9999,
  "capabilities": ["test"],
  "version": "1.0.0"
 }')

AGENT_ID=$(echo "$RESPONSE" | grep -o '"agent_id":"[^"]*"' | cut -d'"' -f4)

if [ -n "$AGENT_ID" ]; then
  echo -e "${GREEN}✅ Agent registered: $AGENT_ID${NC}"
  ((PASSED++))
else
  echo -e "${RED}❌ Agent registration failed${NC}"
  echo "Response: $RESPONSE"
  ((FAILED++))
  exit 1
fi

# 3. List agents
echo ""
echo "3. AGENT DISCOVERY"
curl -s http://localhost:8080/agents | grep -q "$AGENT_ID"
test "Agent appears in list"

# 4. Get specific agent
curl -s http://localhost:8080/agents/$AGENT_ID | grep -q "verify-test-agent"
test "Can retrieve specific agent"

# 5. Send heartbeat
echo ""
echo "4. HEARTBEAT"
curl -s -X POST http://localhost:8080/agents/$AGENT_ID/heartbeat \
 -H "Content-Type: application/json" \
 -d '{"status": "healthy"}' | grep -q "received"
test "Heartbeat accepted"

# 6. List by type
curl -s http://localhost:8080/agents/type/cost | grep -q "$AGENT_ID"
test "Agent found by type filter"

# 7. Unregister
echo ""
echo "5. CLEANUP"
curl -s -X POST http://localhost:8080/agents/$AGENT_ID/unregister | grep -q "successfully"
```

```
test "Agent unregistered"

# Summary
echo ""
echo "========================================================================"
echo "VERIFICATION SUMMARY"
echo "========================================================================"
echo -e "Passed: ${GREEN}$PASSED${NC}"
echo -e "Failed: ${RED}$FAILED${NC}"

if [ $FAILED -eq 0 ]; then
    echo -e "\n${GREEN}✅ ALL TESTS PASSED!${NC}"
    echo "Agent Registry is fully operational!"
    exit 0
else
    echo -e "\n${RED}❌ SOME TESTS FAILED${NC}"
    exit 1
fi
EOF

chmod +x verify_registry.sh
./verify_registry.sh
```

Expected Output:

```
============================================================
FOUNDATION-0.6 COMPREHENSIVE VERIFICATION
============================================================

1. ORCHESTRATOR HEALTH
✅ Orchestrator is healthy

2. AGENT REGISTRATION
✅ Agent registered: abc12345-...

3. AGENT DISCOVERY
✅ Agent appears in list
✅ Can retrieve specific agent

4. HEARTBEAT
✅ Heartbeat accepted
✅ Agent found by type filter

5. CLEANUP
✅ Agent unregistered

============================================================
VERIFICATION SUMMARY
============================================================
Passed: 7
Failed: 0

✅ ALL TESTS PASSED!
Agent Registry is fully operational!
```

🐛 TROUBLESHOOTING

Issue 1: Build Fails - Missing Dependencies

Symptoms:

```
go: github.com/gin-gonic/gin: module not found
```

Solution:

```
bash
```

```
cd ~/optiinfra/services/orchestrator
go mod tidy
go mod download
go build -o bin/orchestrator ./cmd/server
```

Issue 2: Orchestrator Won't Start - Redis Connection

Symptoms:

```
Failed to connect to Redis: connection refused
```

Solution:

```bash
# Check if Redis is running
docker ps | grep redis

# Start Redis if not running
cd ~/optiinfra
docker-compose up -d redis

# Wait and retry
sleep 3
./bin/orchestrator
```

Issue 3: Registration Fails - Port Already in Use

Symptoms:

```
Server failed: listen tcp :8080: bind: address already in use
```

Solution:

```bash
```

```
# Find what's using port 8080
lsof -i :8080

# Kill the process or use different port
export PORT="8081"
./bin/orchestrator
```

Issue 4: Agent Not Found After Registration

Symptoms:

```
curl http://localhost:8080/agents/$AGENT_ID
# Returns: {"error": "Agent not found"}
```

Solution:

```
This happens if:
1. Agent TTL expired (60 seconds in Redis)
2. Redis was restarted
3. Wrong agent ID

Solution:
- Re-register the agent
- Send heartbeats within 30 seconds
- Check Redis: docker exec optiinfra-redis redis-cli KEYS "agent:*"
```

📊 WHAT YOU HAVE NOW

Complete Orchestrator with Agent Registry:

```
OptiInfra Architecture:
├── Databases ✅
│   ├── PostgreSQL (18 tables)
│   ├── ClickHouse (8 tables)
│   ├── Qdrant (3 collections)
│   └── Redis (agent registry)
│
└── Orchestrator (Go) ✅ NEW
    ├── HTTP Server (Gin)
```

```
  ├── Health endpoint
  │
  └── Agent Registry
      ├── Registration API
      ├── Heartbeat monitoring
      ├── Discovery APIs
      ├── Health checker (background)
      └── Redis storage (TTL: 60s)

Total Code: ~1,000 lines (Go + Python)
```

Capabilities Unlocked:

✅ **Agent Management**

- Agents self-register on startup
- Automatic health monitoring
- Discovery by type/capability
- Graceful unregistration

✅ **Monitoring**

- Heartbeat every 30s
- Auto-detect dead agents (45s)
- Background health checker
- Status tracking (healthy/degraded/unreachable)

✅ **Discovery**

- List all agents
- Filter by type
- Filter by capability
- Get only healthy agents

🎊 MILESTONE ACHIEVED

✅ FOUNDATION-0.6 COMPLETE!

You now have:

- ✅ Complete Agent Registry in Go
- ✅ 7 REST APIs for agent management
- ✅ Background health monitoring
- ✅ Redis-backed storage with TTL
- ✅ Python helper for easy integration
- ✅ Comprehensive testing

Foundation Phase Progress:

```
Week 1 Progress: 8/15 prompts (53%)
├── 0.2a: Core Schema ✅
├── 0.2b: Agent State ✅
├── 0.2c: Workflow History ✅
├── 0.2d: Resource Schema ✅
├
```